REMARKS

Claims 1-4, 6-19, 21 and 22 are now pending in this application. Claim 21 is objected to for informal reasons. Claims 16 and 19 stand rejected under 35 U.S.C. § 112, second paragraph as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Claims 1-4 and 6-15 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by United States Patent 6,920,634 ("Tudor") in view A Comparative Analysis of Fine-Grain Threads Packages, Lowenthal et al., 2001 ("Lowenthal"). Claims 16-19 and 21 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Tudor in view of United States Patent 5,193,180 ("Hastings"). Claim 22 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over Tudor in view of Lowenthal and in further view of Hastings. Applicants respectfully traverse.

Claims 1-4, 10-11, 16, 19 and 22 have been amended. Claim 9 is canceled.

Objection Relating To Claim 21

Claim 21 has been objected to as being dependent upon canceled claim 20. Claim 21 has been amended to overcome this limitation and is now made dependent upon claim 1. Thus, the objections to claim 21 should be withdrawn.

Rejection Of Claims 16 and 19 under 35 U.S.C. § 112, second paragraph

Claims 16 and 19 stand rejected under 35 U.S.C. § 112, second paragraph as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Claims 16 and 19 have been amended to overcome these limitation. In particular, claims 16 and 19 have been amended to define instrumentation information as instructions for invoking the race detector. Thus, the rejections of claims 16 and 19 under 35 U.S.C. § 112, second paragraph should be withdrawn.

Rejection Of Claims 1-2 and 5-11 Under 35 U.S.C. § 103(a)

Claims 1-2 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by Tudor in view of Lowenthal. Claim 1 as amended recites a system for dynamically detecting potential race conditions in a program having at least one thread and one or more shared memory locations comprising a processor adapted *to initialize a virtual clock (C_t) respectively for each of the at least one thread, initialize a set of candidate locks for each of at least one shared memory location (S_x), initialize a set of locks held for each of at least one thread (S_t), initialize a set of concurrent thread segments accessing each of the at least one shared memory location (T_x) and initialize an ordered set of thread segments (B_t) below a current thread (t), wherein the ordering is achieved using the virtual clock associated with each thread.*

Claim 1 as amended further *recites upon detection of a fork of a new concurrent thread (t1) by a prior thread t, increment the virtual clock associated with the prior thread (t), initialize the virtual clock associated with the new thread (t1), update the ordered set of thread segments (B_t) below the prior thread t by forming a union of a current ordered set for the prior thread and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time.*

Claim 1 as amended further *recites upon detection of a join call by the prior thread (t) update the ordered set of threads (B_t) by forming a union of the ordered set of thread segments (B_t) a second ordered set of threads (B_t1) below thread t1 that do not belong to the prior thread (t) and a singleton set containing a current thread segment associated with the current thread (t1) and upon detection of a read or write to a memory location (x), determining whether a potential race condition exists.*

Support for this amendment may be found, for example, on pages 14-19. Briefly summarizing this discussion, in addition to recording the per-thread and per-location locksets as in the lockset approach, two further sets are maintained. One set is the set T_x' comprising the set of concurrent thread segments accessing the shared memory location x. A race is reported when the relevant lockset is empty and the cardinality of T_x is greater than 1. The second new set is the set B_t, comprising the set of thread segments ordered before the current thread segment of t. In one embodiment, both T_x and B_t are represented as the set of tuples {<tl, q>, ... ,<tn' cn>}. The ordering relies on the use of a virtual clock Ct for each

thread t. A tuple <t, c> represents the thread segment of thread t at the virtual clock time of c.

Referring to the flowchart of FIG. 7 of the application shows steps associated with the fork of a new concurrent thread tl by a thread t. At step 703, it is determined whether thread t forks thread tl If so, the following operations are performed. At step 703 the set B_tl the set of thread segments ordered before thread tl, is computed as the union of the corresponding set for thread t and the singleton set comprising the thread segment <t, Ct>. At step 705 the virtual clock for thread t is incremented by 1. At step 707 the virtual clock for the newly forked thread tl is initialized to 0. The flowchart of FIG. 7B shows steps associated with the join by thread t. At step 711, it is determined whether thread t makes the join call, waiting for thread tl to complete execution. If so, at step 713, the new value of B_t is computed as the union of the following three sets: (i) B_t, (ii) the set of thread segments in set B_tl that do not belong to the current thread t, and (iii) the singleton set containing the current thread segment of tl.

The flowchart of FIG. 8 of the application shows steps taken when a read or a write of location x is executed by thread t. At step 801 the new value of set T_x is computed, representing the thread segments concurrently accessing location x after the read or write. The second part of the union forming the new T_x is the set containing t's current thread segment <t, Ct>. Since thread t is reading or writing x, clearly t is one of the threads that should be in the set T_x The first part of the union represents a subset of the old value of set T_x in which any thread that is no longer concurrently accessing the location is filtered out. At step 803 it is determined whether the cardinality of the new T_x is less than or equal to 1. If so, there is at most one thread currently accessing x. The new value of S_x then becomes the current value of the lockset S_t (step 805). Otherwise, at step 807, there are multiple concurrent threads accessing x, and the new value of S_x becomes the set comprising the intersection of the old value of S_x and S_t. At step 809 it is determined whether (a) the new value of S_x is empty and (b) the cardinality of the new value of T_x is greater than 1. If so, at step 811 a potential race condition is reported.

In one embodiment, the invention provides a method of analyzing multi-threaded programs. It is determined that unsynchronized accesses to a resource of interest can be performed by a plurality of threads. A request from a first thread to access the resource is received and the first thread 5 is suspended. While the first thread is suspended, a request

from a second thread to access the resource is received. The resource can be a memory location, region of memory, hardware component, peripheral device, and the like.

Referring to FIG. 7B of Tudor, a shared flag of the memory location element is checked at a step 623. If the shared flag is "false" at a step 625 then this is the second thread to access the memory location and the sync object set of the memory location element should be initialized. At a step 629, the shared flag of the memory location element is set to "true." The sync object set of the memory location element is set (or initialized) to include the sync objects held by the current thread at a step 631. The sync object set can be thought of as a set of possible sync objects (or candidate list) that could be being utilized to synchronize access to the memory location. At this time, unless the set of sync objects being held by the thread is the empty set, it is generally unclear if a sync object is being utilized to synchronize access to the memory location. Lockset refinement is utilize to determine if unsynchronized accesses are possible.

Tudor fails to teach or suggest to initializing a virtual clock (C_t) respectively for each of the at least one thread, initialize a set of candidate locks for each of at least one shared memory location (S_x), initialize a set of locks held for each of at least one thread (S_t), initialize a set of concurrent thread segments accessing each of the at least one shared memory location (T_x) and initialize an ordered set of thread segments (B_t) below a current thread (t), wherein the ordering is achieved using the virtual clock associated with each thread. As noted, the application describes two addition sets not described in the lockset approach. Tudor at most addresses the lockset approach and does not disclose in any way the additional sets of T_x and B_t discussed above.

Clearly, since Tudor fails to teach or suggest the T_x or B_T sets, it cannot teach or suggest upon detection of a fork of a new concurrent thread (t1) by a prior thread t, increment the virtual clock associated with the prior thread (t), initialize the virtual clock associated with the new thread (t1), update the ordered set of threads (B_t) below the prior thread t by forming a union of a current ordered set for the prior thread and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time as recited in amended claim 1.

Accordingly, Tudor cannot teach or suggest a join call by the prior thread (t) update the ordered set of threads (B_t) by forming a union of the ordered set of threads (B_t) a

second ordered set of threads (B_t1) below thread t1 that do not belong to the prior thread (t) and a singleton set containing a current thread segment associated with the current thread (t1) and upon detection of a read or write to a memory location (x), determining whether a potential race condition exists as recited in amended claim 1.

Lowenthal fails to cure the deficiencies of Tudor. Further, Lowenthal does not teach or suggest the use of a virtual clock for ordering, associating a virtual clock with each thread or the use of a tuple for referring to thread segments.

As the cited references fail to teach or suggest the discussed claim limitations, claim 1, should be allowed. Claim 2 depends from and therefore includes all the limitations of claim 1. Thus, for at least the reasons stated with respect to claim 1, claim 2 should also be allowed.

Rejection Of Claims 4, 6-8 and 10-15 Under 35 U.S.C. § 103(a)

Claims 4, 6-8 and 10-15 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by Tudor in view of Lowenthal. Independent claim 4 recites limitations similar to amended claim 1 including *wherein the thread segments are ordered using each associated respective virtual clock and upon detection of a fork of a new concurrent thread (t1) by a prior thread t incrementing the virtual clock associated with the prior thread (t), initializing the virtual clock associated with the new thread (t1), updating the ordered set of threads (B_t) below the prior thread t by forming a union of a current ordered set for the prior thread and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time and upon detection of a read or write to a memory location (x), determining whether a potential race condition exists.*

Claims 6-8 and 10-15 depends from and therefore includes all the limitations of claim 4. Thus, for at least the reasons stated with respect to claim 4, claims 6-8 and 10-15 should also be allowed.

Rejection Of Claims 16-19 and 21 and 22 Under 35 U.S.C. § 103(a)

Claims 16-19 and 21 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by Tudor in view of Hastings. Independent claim 16 as amended recites a dynamic race detection system, comprising a hardware means for modifying and

implementing a compiler of a runtime system that inserts calls to a race detector in compiled code, wherein the calls to the race detector are dynamically inserted by a JIT ("Just-In-Time") compiler and a hardware means for implementing a memory allocator of the runtime system that adds to shared memory objects instrumentation information required by the race detector, wherein the instrumentation information indicates instructions for invoking the race detector.

Hastings fails to teach or suggest dynamic insertion using a JIT. Claims 17-18 depend from and therefore includes all the limitations of claim 16. Thus, for at least the reasons stated with respect to claim 16, claims 17-18 should also be allowed.

Claim 19 as amended includes limitations similar to claim 16 relating to a JIT. Thus, for at least the reasons stated with respect to claim 16, claim 19 should be allowed. Claim Claims 21 depends from and therefore includes all the limitations of claim 19. Thus, for at least the reasons stated with respect to claim 19, claim 21 should also be allowed.

Claim 22 as amended includes limitations similar to claim 16 and 19 relating to a JIT. Thus, for at least the reasons stated with respect to claim 16 and claim 19, claim 22 should be allowed.

CONCLUSION

In view of the above amendments and remarks, applicant respectfully submits that the present invention is in condition for allowance. Reconsideration of the application is respectfully requested.

Date: December 1, 2008

/Kenneth R. Eiferman/
Kenneth R. Eiferman
Registration No. 51,647

Woodcock Washburn LLP
Cira Centre
2929 Arch Street, 12th Floor
Philadelphia, PA 19104-2891
Telephone: (215) 568-3100
Facsimile: (215) 568-3439